

# On the Computational Complexity of Satisfiability Solving for String Theories

Susmit Jha      Sanjit A. Seshia      Rhishikesh Limaye  
EECS Department, UC Berkeley  
{jha,sseshia,rhishi}@eecs.berkeley.edu

March 16, 2009

## Abstract

Satisfiability solvers are increasingly playing a key role in software verification, with particularly effective use in the analysis of security vulnerabilities. String processing is a key part of many software applications, such as browsers and web servers. These applications are susceptible to attacks through malicious data received over network. Automated tools for analyzing the security of such applications, thus need to reason about strings. For efficiency reasons, it is desirable to have a solver that treats strings as first-class types. In this paper, we present some theories of strings that are useful in a software security context and analyze the computational complexity of the presented theories. We use this complexity analysis to motivate a byte-blast approach which employs a Boolean encoding of the string constraints to a corresponding Boolean satisfiability problem.

## 1 Introduction

Many security-critical applications such as Web servers routinely process strings as an essential part of their functionality. They take strings as inputs, screen them using filters, manipulate them and use them for operations such as database queries. It is pertinent to verify that these programs do not have vulnerabilities which can be used to compromise system security. Verification and structured testing techniques to validate security of such applications often rely on using constraint solvers. The frequent use of string operations in these applications has motivated several groups to explore the possibility of designing a constraint solver which treats strings as first-class types. Such a specialized solver for strings would further facilitate the use of constraint solving for analysis of security applications with string operations.

Software applications use various string predicates and functions which are often made available to the developers as libraries. A satisfiability solver for string constraints must be able to handle these predicates and functions. From the string constraints and predicates available in high level programming languages such as C, JAVA and C++, we identify a set of core predicates and functions. Many other more complicated string-manipulating functions can be expressed as some simple composition of these functions. We use these predicates and functions to define a theory of strings.

The main contribution of this paper is an analysis of the complexity of several fragments of the theory of strings. We show that fairly small and simple-looking fragments are NP-complete. In light of the progress in SAT solving and SMT solving for bit-vector arithmetic, these results indicate that a SAT-based approach is reasonable to satisfiability solving of string constraints.

## 2 Related Work

Constraint solvers are widely used in verification and validation of software and hardware systems [7, 16, 26]. In particular, they have been used extensively for both static [10, 4, 3] and dynamic analysis [13, 15] of programs to detect malicious code or security vulnerabilities in benign code. The use of constraint solving in software verification is driven by development of faster and more scalable SMT solvers for the theory of bit-vectors such as BAT [22], Boolector [5], Beaver [19] MathSat [6], Spear [18], STP [14], UCLID [7] and Z3 [12]. In particular, UCLID and STP have been successfully used for security applications. For example, bit-vector solvers can be used to easily detect overflow/underflow errors which are cause of many security vulnerabilities such as buffer overflow [24].

Analysis of string processing software is an important problem [27, 25, 28]. This makes it essential to develop verification techniques that can efficiently handle constraints over strings. A scalable approach for solving string constraints must treat strings as first class types and string library functions as native operations of the theory of strings [8]. Development of such a solver for a theory of strings would further facilitate the use of constraint solving for program analysis, in general, and security applications, in particular. This will further push the frontier of program analysis in terms of scalability as well as program complexity.

While previous efforts have been made to develop decision procedures for regular expression containment [17, 9], there have been some recent efforts to develop an SMT solver for the theory of strings.

In an independent and parallel work, Kiezun et al [20] have developed a solver (HAMPI) for a theory of strings. HAMPI works by reducing the formulae over string constraints to bit-vector logic and then, using a bit-vector solver (STP) for checking the satisfiability of the formulae. This reduction is achieved in two steps. HAMPI reduces the string constraints specified using a rich input language to a core theory of strings comprising of regular language operations and membership predicate. The string constraints in this core language are then translated to bit-vector logic before invoking a bit-vector solver. They also show that the satisfiability problem for this theory of strings with regular expression operations is NP-complete.

The string theory considered in this paper is different from the one considered in HAMPI. The string functions and predicates in our theory of strings are motivated by commonly used library functions in high level languages such as C and Java. The set of constraints expressible in our theory of strings are not comparable with the set of constraints expressible in HAMPI. We identify constraints which can be expressed in our theory and not in HAMPI as well as those which can be expressed in HAMPI but not in our theory.

1. Our theory has *contains-at-position-i* predicate which is true if and only if its first argument string is contained in its second argument string at exactly position *i*. We also have *extract-i-j* function which extracts a sub-string from its string argument using the indices *i* and *j*. While the SMT solving approach of HAMPI can be used to handle these constraints, the theory of strings considered in HAMPI is based on regular languages and can not be used to encode these constraints.
2. Our theory does not have union or star operation and hence, constraints with union or star can not be expressed in our theory.

In particular, we note that the NP-completeness result established in Kiezun [20] relies on the use of union operation to provide disjunction. *We show that even without this union operation, the theory of strings is NP-complete and all non-trivial fragments of the theory of strings are also NP-complete.*

Bjorner et al [2] propose another approach to solving string constraints arising out of path feasibility queries. Their approach relies on identifying candidate string lengths by solving length constraints and then, solving the string constraints by considering them to be of lengths found in the first phase. The string lengths found in the first phase may not provide a solution even if the formulae is satisfiable and hence, they need to iterate with different length assignments. The string operations considered in this work are similar to ones proposed here. We consider strings of bounded lengths and we do not consider the *replace* operation. Hence, our fragment of string theory is decidable. *In contrast to Bjorner et al's work who presented decidability result for theory of strings, we present complexity results on the theory of strings and its different fragments.*

### 3 Theory Definition

The definition of the theory of strings presented in this section is motivated by checking path feasibility queries over programs written in some high level languages such as Java, C and Ocaml. The string libraries used in these high level languages are abstracted as string functions and predicates. We now define the complete theory of strings using these predicates and functions in this section. Later, we will analyze the complexity of different fragments of this theory by considering different subset of string predicates and functions.

$$\begin{aligned}
str\text{-}expr &::= c \mid s \mid str\text{-}expr[i : j] \mid str\text{-}expr_1 @ str\text{-}expr_2 \\
bool\text{-}expr &::= \mathbf{true} \mid \mathbf{false} \mid \neg bool\text{-}expr \\
&\quad \mid str\text{-}expr_1 = str\text{-}expr_2 \mid str\text{-}expr_1 \sqsupseteq str\text{-}expr_2 \mid str\text{-}expr_1 \sqsupseteq_i str\text{-}expr_2 \\
formula &::= bool\text{-}expr \mid bool\text{-}expr \wedge formula \\
i, j &\in \mathbb{N} \quad s, s_i \text{ are string variables} \quad c \text{ represents a string constant.}
\end{aligned}$$

Figure 1: **Syntax for String Logic**  $[i : j]$  denotes extraction of the sub-string starting at position  $i$  and ending at position  $j$ ;  $@$  denotes concatenation;  $\sqsupseteq$  denotes containment; and  $\sqsupseteq_i$  denotes containment at position  $i$ .

The syntax of the statements in theory of strings is given by grammar in Figure 1. The strings are over some finite alphabet  $\Sigma$ . The string constraints arising from software verification involve only finite length strings. The length of a string is bounded by the length of the corresponding buffer. So, we require that the maximum length of each string is bounded by a constant. Also, the maximum length of all strings are less than some constant  $L_{max}$ . Also, there is an empty string constant  $\epsilon$ . We describe the semantics of the predicates and functions used in the theory definition below.

**String Predicates:** The string predicates take two string arguments and evaluate to true or false.

1. Equality:  $s_i = s_j$  is true if and only if both  $s_i$  and  $s_j$  are assigned the same string constants and otherwise, it is false.
2. Containment at position  $i$ :  $s_1 \sqsupseteq_i s_2$  denotes that  $s_2$  is contained in  $s_1$  at position  $i$ . For example, *bombay* contains *bay* at location 4. So, *bombay*  $\sqsupseteq_4$  *bay* evaluates to true.
3. Containment:  $s_1 \sqsupseteq s_2$  denotes that  $s_2$  is contained in  $s_1$  at some position. In particular, the empty string  $\epsilon$  is contained in all strings and does not contain any non-empty string, that is,  $\forall s, s \sqsupseteq \epsilon$  and  $\forall s, s \neq \epsilon \Rightarrow \neg(\epsilon \sqsupseteq s)$ .

**String Functions:** The two string functions considered in this paper are extraction and concatenation.

1. Extraction:  $s[i : j]$  has the type signature  $str\text{-}expr \times int \times int \rightarrow str\text{-}expr$ . It denotes the substring of  $s$  starting from position  $i$  and ending at position  $j$  where  $i$  and  $j$  are integers. For example, *bombay* $[4 : 6]$  evaluates to *bay*.
2. Concatenation:  $s_1 @ s_2$  has the type signature  $str\text{-}expr \times str\text{-}expr \rightarrow str\text{-}expr$ . It denotes the concatenation of the two strings provided to it as arguments. For example, *bom*@*bay* evaluates to *bombay*.

### 4 Complexity Results

Before stating and proving the complexity results, we present a brief summary of the results in this section and note that all non-trivial fragments of theory of strings are NP-complete. In Theorem 1, we show that the satisfiability problem for the theory of strings as define in Section 3 is in NP. This is a direct consequence of having a constant bound on the

size of any string. Hence, the satisfiability of any fragment of string theory is also in NP. Each fragment of theory of strings is defined by selecting some string predicates and functions along with Boolean negation and conjunction. As discussed in Section 3, there are two functions and three predicates. To define a fragment of string theory we need to include atleast one string predicate.

The three most elementary fragment of string theory are defined by including exactly one string predicate.

1. E: This fragment consists of string equality, Boolean negation and conjunction.
2. C: This fragment consists of string containment, Boolean negation and conjunction.
3. T: This fragment consists of string containment at position  $i$ , Boolean negation and conjunction.

It is shown that the satisfiability problem for C fragment is NP-complete in Theorem 6. The satisfiability problem for T fragment is also NP-complete as shown in Lemma 2. We know that E fragment is polynomial-time solvable using congruence closure [1]. So, we extend the E (equality) fragment with different string predicates and functions, and analyze its complexity.

1. E+C: This fragment extends E with string containment.
2. E+T: This fragment extends E with string containment at position  $i$ . If  $i$  is only allowed to be constant, the corresponding logic is E+T-CONST.
3. E+A: This fragment extends E with string concat function.
4. E+X: This fragment extends E with string extract function. If the indices for extract are only constant, the corresponding logic is E+X-CONST.

Since the satisfiability problems for C and T fragments are NP-complete, it is natural that E+C and E+T would also be NP-complete. We have separately proved the hardness results for both fragments in Theorem 2 and Theorem 3. It is also shown in Theorem 4 and Theorem 5 that the satisfiability problem for E+A and E+X fragments are also NP complete.

Any extension of these fragments would also be NP-complete. So, the NP-completeness results for these minimal fragments of string theory presented in this section imply that the satisfiability problem for all fragments of string theory except for the E (equality) fragment is computationally hard. Thus, it is unlikely that there is any polynomial time algorithm for deciding the satisfiability of any non-trivial fragment of string theory unless  $P=NP$ .

In the rest of the section, we state and prove the complexity results.

**Theorem 1** *The satisfiability problem over the theory of string is in NP.*

*Proof:* If the formula over theory of strings is satisfiable, then the satisfying instance is an assignment of string variables to strings with lengths upper bounded by the constant  $L_{max}$ . Hence, the size of the satisfying assignment is at most  $L_{max}N$  where  $N$  is the number of string variables. So, the length of the certificate is polynomial in the size of the input and hence, satisfiability of formula in theory of strings is in NP.  $\square$

As a consequence of the above theorem, the satisfiability of formulae in smaller fragments of theory of strings such as E+C, E+T-CONST, E+A and E+X-CONST is also in NP. Hence, we only require to show that satisfiability of formulae in these fragments is NP-hard in order to prove that the satisfiability problem for these fragments is NP-complete.

In rest of the section, we state and prove the NP-hardness results for each of these fragments. We show that the satisfiability problem for different fragments of string theory is NP-hard. Let us consider a 3-CNF formula  $\phi$  over a set  $X = \{x_1, x_2, \dots, x_n\}$  of  $n$  Boolean variables.

$$\phi \equiv \bigwedge_{i=1}^m (l_1^i \vee l_2^i \vee l_3^i)$$

where each literal  $l_i^j$  is  $x_k$  or  $\neg x_k$  for some  $x_k \in X$ . We know that 3-CNF-SAT is NP-complete [11]. We now reduce this problem, that is, finding an assignment of variables in  $X$  to  $\{0, 1\}$  such that  $\phi$  evaluates to 1, to the problem of

finding a satisfying assignment in the corresponding fragment of theory of strings.

#### 4.1 Equality + Containment (E+C)

**Theorem 2** *The satisfiability problem over the theory of strings with equality, contains, Boolean negation and conjunction (E+C fragment) is NP-hard.*

*Proof:* We prove this by reducing 3-CNF-SAT to E+C fragment of theory of strings. We describe a transformation that maps a 3-CNF Boolean formula to a formula in E+C fragment of theory of strings (over the alphabet  $\Sigma = \{a\}$ ) such that there is a satisfying assignment for the Boolean formula if and only if there is a satisfying assignment for the formula over strings.

Let  $\psi$  be defined as

$$\psi(x_i) \triangleq s_i \text{ and } \psi(\neg x_i) \triangleq r_i$$

where  $s_i$  and  $r_i$  are strings of atmost length 1.  $s_i = a$  if and only if  $x_i$  is assigned true, otherwise, it is  $\epsilon$ . Similarly,  $r_i = a$  if and only if  $x_i$  is assigned false, otherwise it is  $\epsilon$ . So, for any literal  $l$ ,  $\psi(l)$  would be  $a$  if and only if  $l$  is assigned true.

We also need to add constraints to ensure consistency, that is, exactly one of  $x_i$  or  $\neg x_i$  is assigned true. For consistency, for each variable  $x_i$ , we must have the constraint

$$s_i \neq r_i$$

This ensures that exactly one of  $s_i$  or  $r_i$  is  $a$ .

Each clause  $c \equiv l_1 \vee l_2 \vee l_3$  is transformed to

$$V_c \supseteq \psi(l_1)$$

$$V_c \supseteq \psi(l_2)$$

$$V_c \supseteq \psi(l_3)$$

$$V_c \neq \epsilon$$

where  $V_c$  is a new string variable for clause  $c$  and is of length atmost 3.

Thus, atleast one of  $\psi(l_i)$  must be  $a$  which is possible if and only  $l_i$  is assigned true. So, atleast one literal in each clause is true.

A set of string constraints  $\psi(\phi)$  is obtained by applying the above transformations to each clause  $c_i$  in 3-CNF Boolean formula  $\phi$  and taking the union of all the obtained string constraints.

Let  $\mathcal{I}$  be a satisfying assignment to  $\phi$  such that  $\mathcal{I}(x)$  denotes the assignment to  $x$ . By construction, there is an assignment  $\mathcal{I}'$  to  $\psi(\phi)$  such that  $\mathcal{I}'(s_i) = a$  and  $\mathcal{I}'(r_i) = \epsilon$  if and only if  $\mathcal{I}(x_i) = \text{true}$ .

Thus, E+C fragment of string theory is NP-hard.  $\square$

**Corollary 1** *The satisfiability problem over the theory of strings with equality, contains at  $i$  where  $i$  is variable, Boolean negation and conjunction (E+T-VAR fragment) is NP-hard.*

*Proof:*  $str_1 \supseteq str_2$  can be rewritten as  $str_1 \supseteq_i str_2$  where  $i$  is a new index variable. Hence, any formula in E+C can be expressed as a formula in E+T-VAR fragment.  $\square$

## 4.2 Equality + Containment-aT-Constant (E+T-CONST)

**Theorem 3** *The satisfiability problem over the theory of strings with contains at constant position, equality, Boolean negation and conjunction (E+T-CONST fragment) is NP-hard.*

*Proof:* We prove this by reducing 3-CNF-SAT to E+T-CONST fragment of theory of strings. We describe a transformation that maps a 3-CNF Boolean formula to a formula in E+T-CONST fragment of theory of strings (over the alphabet  $\Sigma = \{a, b\}$ ) such that there is a satisfying assignment for the Boolean formula if and only if there is a satisfying assignment for the formula over strings.

Let  $\psi$  be defined as

$$\psi(x_i) \triangleq s_i \text{ and } \psi(\neg x_i) \triangleq r_i$$

where  $s_i$  and  $r_i$  are strings of atmost length 1. To make it exactly of length 1, we require  $s_i \neq \epsilon \wedge r_i \neq \epsilon$ .  $s_i = a$  if and only if  $x_i$  is assigned true, otherwise, it is  $b$ . Similarly,  $r_i = a$  if and only if  $x_i$  is assigned false, otherwise it is  $b$ . So, for any literal  $l$ ,  $\psi(l)$  would be  $a$  if and only if  $l$  is assigned true.

We also need to add constraints to ensure consistency, that is, exactly one of  $x_i$  or  $\neg x_i$  is assigned true. For consistency, for each variable  $x_i$ , we must have the constraint

$$s_i \neq r_i$$

This ensures that exactly one of  $s_i$  or  $r_i$  is  $a$ .

Each clause  $c \equiv l_1 \vee l_2 \vee l_3$  is transformed to

$$V_c \supseteq_1 \psi(l_1)$$

$$V_c \supseteq_2 \psi(l_2)$$

$$V_c \supseteq_3 \psi(l_3)$$

$$V_c \neq bbb$$

where  $V_c$  is a new variable for clause  $c$  and is of length atmost 3.

Thus, atleast one of  $\psi(l_i)$  must be of  $a$  which is possible if and only if  $l_i$  is assigned true. So, atleast one literal in each clause is true.

A set of string constraints  $\psi(\phi)$  is obtained by applying the above transformations to each clause  $c_i$  in 3-CNF Boolean formula  $\phi$  and taking the union of all the obtained string constraints.

Let  $\mathcal{I}$  be a satisfying assignment to  $\phi$  such that  $\mathcal{I}(x)$  denotes the assignment to  $x$ . By construction, there is an assignment  $\mathcal{I}'$  to  $\psi(\phi)$  such that  $\mathcal{I}'(s_i) = a$  and  $\mathcal{I}'(r_i) = b$  if and only if  $\mathcal{I}(x_i) = \text{true}$ .

Thus, E+T-CONST fragment of string theory is NP-hard.  $\square$

**Corollary 2** *The satisfiability problem over the theory of strings with contains at constant position, Boolean negation and conjunction is NP-hard.*

*Proof:* In the proof above, we can replace  $s_i \neq r_i$  by  $\neg(s_i \supseteq_1 r_i)$  and  $V_c \neq bbb$  by  $\neg(V_c \supseteq_1 bbbb)$ . The NP-hardness proof still goes through. Dis-equality between the strings of same length can be expressed as dis-containment-at-1.  $\square$

## 4.3 Equality + concAt (E+A)

**Theorem 4** *The satisfiability problem over the theory of strings with equality, concat and Boolean conjunction (E+A fragment) is NP-hard.*

*Proof:* We prove this by reducing 3-CNF-SAT to E+A fragment of theory of strings. We describe a transformation that maps a 3-CNF Boolean formula to a formula in E+A fragment of theory of strings (over the alphabet  $\Sigma = \{a\}$ ) such that there is a satisfying assignment for the Boolean formula if and only if there is a satisfying assignment for the formula over strings.

Let  $\psi$  be defined as

$$\psi(x_i) \triangleq s_i \text{ and } \psi(\neg x_i) \triangleq r_i$$

where  $s_i$  and  $r_i$  are strings of atmost length 1.  $s_i = a$  if and only if  $x_i$  is assigned true, otherwise, it is  $\epsilon$ . Similarly,  $r_i = a$  if and only if  $x_i$  is assigned false, otherwise it is  $\epsilon$ . So, for any literal  $l$ ,  $\psi(l)$  would be  $a$  if and only if  $l$  is assigned true.

We also need to add constraints to ensure consistency, that is, exactly one of  $x_i$  or  $\neg x_i$  is assigned true. For consistency, for each variable  $x_i$ , we must have the constraint

$$s_i @ r_i = a$$

This ensures that exactly one of  $s_i$  or  $r_i$  is  $a$ , that is, exactly one of  $\psi(x_i)$  or  $\psi(\neg x_i)$  is  $a$ .

Each clause  $l_1 \vee l_2 \vee l_3$  is transformed to

$$\psi(l_1) @ \psi(l_2) @ \psi(l_3) @ p_i = aaa$$

where  $p_i$  is of length atmost 2. Thus, the sum of the lengths of  $\psi(l_1)$ ,  $\psi(l_2)$  and  $\psi(l_3)$  must be atleast 1, that is, atleast one of  $\psi(l_i)$  must be  $a$  which is possible if and only  $l_i$  is assigned true. So, atleast one literal in each clause is true.

A set of string constraints  $\psi(\phi)$  is obtained by applying the above transformations to each clause  $c_i$  in 3-CNF Boolean formula  $\phi$  and taking the union of all the obtained string constraints.

Let  $\mathcal{I}$  be a satisfying assignment to  $\phi$  such that  $\mathcal{I}(x)$  denotes the assignment to  $x$ . By construction, there is an assignment  $\mathcal{I}'$  to  $\psi(\phi)$  such that  $\mathcal{I}'(s_i) = a$  and  $\mathcal{I}'(r_i) = \epsilon$  if and only if  $\mathcal{I}(x_i) = \text{true}$ .

Thus, E+A fragment of string theory is NP-hard.

□

**Corollary 3** *The satisfiability problem over the theory of strings with contains ( $\sqsubseteq$ ) and concat (C+A fragment) is NP-hard.*

*Proof:* Equality can be expressed with two-way containment. Once again, note that there is no negation in this fragment. □

#### 4.4 Equality + eXtract-with-constant-indices (E+X-Const)

**Theorem 5** *The satisfiability problem over the theory of strings with equality, extract with constant indices, Boolean negation and conjunction (E+X-CONST fragment) is NP-hard.*

*Proof:* We prove this by reducing 3-CNF-SAT to E+X-CONST fragment of theory of strings. We describe a transformation that maps a 3-CNF Boolean formula to a formula in E+X-CONST fragment of theory of strings (over the alphabet  $\Sigma = \{a, b\}$ ) such that there is a satisfying assignment for the Boolean formula if and only if there is a satisfying assignment for the formula over strings.

Let  $\psi$  be defined as

$$\psi(x_i) \triangleq s_i \text{ and } \psi(\neg x_i) \triangleq r_i$$

where  $s_i$  and  $r_i$  are strings of atmost length 1. To make it exactly of length 1, we require  $s_i \neq \epsilon \wedge r_i \neq \epsilon$ .  $s_i = a$  if and only if  $x_i$  is assigned true, otherwise, it is  $b$ . Similarly,  $r_i = a$  if and only if  $x_i$  is assigned false, otherwise it is  $b$ . So, for any literal  $l$ ,  $\psi(l)$  would be  $a$  if and only if  $l$  is assigned true.



We also need to add constraints to ensure consistency, that is, exactly one of  $x_i$  or  $\neg x_i$  is assigned true. For consistency, for each variable  $x_i$ , we must have the constraint

$$s_i \neq r_i$$

This ensures that exactly one of  $s_i$  or  $r_i$  is  $a$ .

Each clause  $c \equiv l_1 \vee l_2 \vee l_3$  is transformed to

$$V_c[1 : 1] = \psi(l_1)$$

$$V_c[2 : 2] = \psi(l_2)$$

$$V_c[3 : 3] = \psi(l_3)$$

$$V_c \neq bbb$$

where  $V_c$  is a new variable for clause  $c$  and is of length atmost 3.

Thus, atleast one of  $\psi(l_i)$  must be of  $a$  which is possible if and only  $l_i$  is assigned true. So, atleast one literal in each clause is true.

A set of string constraints  $\psi(\phi)$  is obtained by applying the above transformations to each clause  $c_i$  in 3-CNF Boolean formula  $\phi$  and taking the union of all the obtained string constraints.

Let  $\mathcal{I}$  be a satisfying assignment to  $\phi$  such that  $\mathcal{I}(x)$  denotes the assignment to  $x$ . By construction, there is an assignment  $\mathcal{I}'$  to  $\psi(\phi)$  such that  $\mathcal{I}'(s_i) = a$  and  $\mathcal{I}'(r_i) = b$  if and only if  $\mathcal{I}(x_i) = \text{true}$ .

Thus, E+X-CONST fragment of string theory is NP-hard.

□

We now show that even without equality, the fragment of the theory of strings having *contains* as string predicate with Boolean negation and conjunction is also hard. This is the final result of the section.

## 4.5 Containment (C)

**Theorem 6** *The satisfiability problem over the theory of strings with contains, Boolean negation and conjunction (C fragment) is NP-hard.*

*Proof:* We prove this by reducing 3-CNF-SAT to C fragment of theory of strings. We describe a transformation that maps a 3-CNF Boolean formula to a formula in C fragment of theory of strings (over the alphabet  $\Sigma = \{a, b\}$ ) such that there is a satisfying assignment for the Boolean formula if and only if there is a satisfying assignment for the formula over strings.

Let  $\psi$  be defined as

$$\psi(x_i) \triangleq s_i \text{ and } \psi(\neg x_i) \triangleq r_i$$

where  $s_i$  and  $r_i$  are strings of atmost length 1. To make it exactly of length 1, we require  $\neg(\epsilon \sqsupseteq s_i) \wedge \neg(\epsilon \sqsupseteq r_i)$ .  $s_i = a$  if and only if  $x_i$  is assigned true, otherwise, it is  $b$ . Similarly,  $r_i = a$  if and only if  $x_i$  is assigned false, otherwise it is  $b$ . So, for any literal  $l$ ,  $\psi(l)$  would be  $a$  if and only if  $l$  is assigned true.

We also need to add constraints to ensure consistency, that is, exactly one of  $x_i$  or  $\neg x_i$  is assigned true. For consistency, for each variable  $x_i$ , we must have the constraint

$$\neg(s_i \sqsupseteq r_i)$$

Each clause  $c \equiv l_1 \vee l_2 \vee l_3$  is transformed to

$$V_c \sqsupseteq \psi(l_1)$$

$$V_c \sqsupseteq \psi(l_2)$$



$$V_c \sqsupseteq \psi(l_3)$$

$$\neg(bbb \sqsupseteq V_c)$$

where  $V_c$  is a new variable for clause  $c$  and is of length atmost 3.

Thus, atleast one of  $\psi(l_i)$  must be of  $a$  which is possible if and only  $l_i$  is assigned true. So, atleast one literal in each clause is true.

A set of string constraints  $\psi(\phi)$  is obtained by applying the above transformations to each clause  $c_i$  in 3-CNF Boolean formula  $\phi$  and taking the union of all the obtained string constraints.

Let  $\mathcal{I}$  be a satisfying assignment to  $\phi$  such that  $\mathcal{I}(x)$  denotes the assignment to  $x$ . By construction, there is an assignment  $\mathcal{I}'$  to  $\psi(\phi)$  such that  $\mathcal{I}'(s_i) = a$  and  $\mathcal{I}'(r_i) = b$  if and only if  $\mathcal{I}(x_i) = \text{true}$ .

Thus, C fragment of string theory is NP-hard.  $\square$

## 5 Conclusion and Future Work

The analysis of different fragments of the theory of strings presented in this paper shows that the satisfiability problem for even small non-trivial fragments is NP-complete. Thus, it is unlikely that an efficient (polynomial-time) algorithm for checking the satisfiability of the strings would be found. Hence, a simple approach based on Boolean encoding of string constraints to propositional logic is, in principle, as effective as any other technique for solving string constraints. This justifies a “byte-blast” approach to solving string constraints which relies on encoding strings as bit-vectors and using an off-the-shelf bit-vector SMT solver. Further, these hardness results underline the importance of using domain knowledge about string constraints arising out of security applications. We believe, in practice, word-level reasoning over strings that exploits such domain knowledge through pragmatic approaches such as abstraction-refinement might prove to be very effective in making an efficient and scalable for theory of strings. The key challenge in developing an SMT solver for theory of strings is identification of such properties of string constraints arising from real code.

Inspired by the success of abstraction-refinement based approaches for SMT solving (e.g., [21, 7, 14]), we believe such an approach would be useful for the theory of strings also. We identify the abstraction techniques that we believe would be especially useful in the context of a theory of strings:

1. *Length abstraction:* To our knowledge, this approach has been first published by Bjorner et al [2]. It operates by creating an over-approximation of the actual formula by abstracting each string constraint with a corresponding length constraint. The resulting integer linear arithmetic formula is solved to obtain candidate lengths for the strings in the original formula, with a possible refinement needed if these candidate lengths turn out to be too small. We believe that this general idea can be used but with some guidance to the solver to not simply generate the smallest lengths.
2. *Position abstraction:* We have observed that, in the security applications of interest, *string-containment* is a widely used predicate and the encoding the choice of position of containment adds significant complexity to the constraint satisfaction problem. For large string-lengths, a standard byte-blast approach which reduces the string constraints to bit-vector formula would require the SAT solver to branch over a large set of choices of positions. We hypothesize based on our observations of string constraints generated by colleagues in security applications [8], that the position and order of containment of substrings is often not critical to finding a satisfying assignment. Hence, an effective approach to construct under-approximation of the string formula would be fixing some heuristic ordering of containment constraints. If the formula with this fixed ordering is unsatisfiable, the unsat core generated by the SAT solver can be used to selectively refine the ordering.

The overall approach we envisage will be similar to the iterative construction of over- and under-approximate formulas as performed in prior work on model checking [23] and SMT solving for bit-vector arithmetic [7]. It would be interesting to evaluate how such an approach based on abstraction-refinement performs for string formulas generated in practice from security applications.

## Acknowledgments

We are grateful to Juan Caballero and Dawn Song for many helpful discussions on formalizing fragments of the theory of strings that are relevant for software security.

## References

- [1] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 4, chapter 8. IOS Press, 2009.
- [2] N. Bjorner, N. Tillmann, and A. Voronkov. Path feasibility analysis for string-manipulating programs. Technical Report MSR-TR-2008-153, Microsoft Research, 2008.
- [3] D. Brumley, J. Caballero, Z. Liang, J. Newsome, and D. Song. Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation. In *In Proceedings of the 16th USENIX Security Symposium (Security07)*, 2007.
- [4] D. Brumley, H. Wang, S. Jha, and D. Song. Creating vulnerability signatures using weakest preconditions. *Computer Security Foundations Symposium, 2007. CSF '07. 20th IEEE*, pages 311–325, July 2007.
- [5] R. D. Brummayer and A. Biere. Boolector: An efficient SMT solver for bit-vectors and arrays. In *Proc. of TACAS*, 2009. To appear.
- [6] R. Bruttomesso, A. Cimatti, A. Franzen, A. Griggio, Z. Hanna, A. Nadel, A. Palti, and R. Sebastiani. A lazy and layered SMT(BV) solver for hard industrial verification problems. In *Computer Aided Verification (CAV)*, LNCS 4590, pages 547–560, 2007.
- [7] R. E. Bryant, D. Kroening, J. Ouaknine, S. A. Seshia, O. Strichman, and B. Brady. Deciding bit-vector arithmetic with abstraction. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *Lecture Notes in Computer Science*, pages 358–372. Springer, 2007.
- [8] J. Caballero, S. McCamant, A. Barth, and D. Song. Extracting models of security-sensitive operations using string-enhanced white-box exploration on binaries. Technical Report UCB/EECS-2009-36, EECS Department, University of California, Berkeley, March 2009.
- [9] A. S. Christensen, A. Møller, and M. I. Schwartzbach. Precise analysis of string expressions. In *Proc. 10th International Static Analysis Symposium, SAS '03*, volume 2694 of *LNCS*, pages 1–18. Springer-Verlag, June 2003. Available from <http://www.brics.dk/JSA/>.
- [10] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-aware malware detection. *Security and Privacy, 2005 IEEE Symposium on*, pages 32–46, May 2005.
- [11] S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.
- [12] L. de Moura and N. Bjorner. Z3: An efficient SMT solver. In *Proc. TACAS*, 2008.
- [13] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song. Dynamic spyware analysis. In *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2007. USENIX Association.
- [14] V. Ganesh and D. L. Dill. A decision procedure for bit-vectors and arrays. In *Computer Aided Verification (CAV '07)*, Berlin, Germany, July 2007. Springer-Verlag.

- [15] P. Godefroid, M. Y. Levin, and D. Molnar. Automated Whitebox Fuzz Testing. In *Proceedings of the Network and Distributed System Security Symposium*, 2008.
- [16] S. Gulwani, S. Srivastava, and R. Venkatesan. Program analysis as constraint solving. *SIGPLAN Not.*, 43(6):281–292, 2008.
- [17] P. Hooimeijer and W. Weimer. A decision procedure for subset constraints over regular languages. *PLDI (to appear)*, 2009.
- [18] F. Hutter, D. Babic, H. H. Hoos, and A. J. Hu. Boosting verification by automatic tuning of decision procedures. In *7th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 27–34. IEEE Computer Society, 2007.
- [19] S. Jha, R. Limaye, and S. A. Seshia. Beaver - a bit-vector SMT solver. <http://uclid.eecs.berkeley.edu/Beaver>.
- [20] A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst. HAMPI: A solver for string constraints. Technical Report MIT-CSAIL-TR-2009-004, MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, February 4, 2009.
- [21] D. Kroening, J. Ouaknine, S. A. Seshia, and O. Strichman. Abstraction-based satisfiability solving of Presburger arithmetic. In *Proc. 16th International Conference on Computer-Aided Verification (CAV)*, pages 308–320, July 2004.
- [22] P. Manolios, S. K. Srinivasan, and D. Vroon. BAT: the bit-level analysis tool. In *Computer Aided Verification (CAV '07)*, pages 303–306, Berlin, Germany, July 2007. Springer-Verlag.
- [23] K. McMillan and N. Amla. Automatic abstraction without counterexamples. In H. Garavel and J. Hatcliff, editors, *TACAS'03*, volume 2619 of *LNCS*, 2003.
- [24] D. A. Molnar and D. Wagner. Catchconv: Symbolic execution and run-time type inference for integer conversion errors. Technical Report 2007-23, University of California Berkeley, February 2007.
- [25] H. Ruan, J. Zhang, and J. Yan. Test data generation for c programs with string-handling functions. *Theoretical Aspects of Software Engineering, 2008. TASE '08. 2nd IFIP/IEEE International Symposium on*, pages 219–226, June 2008.
- [26] A. Tiwari and S. Gulwani. Logical interpretation: Static program analysis using theorem proving. In F. Pfenning, editor, *CADE-21*, volume 4603 of *LNAI*, pages 147–166. Springer, 2007.
- [27] G. Wassermann and Z. Su. Sound and precise analysis of web applications for injection vulnerabilities. *SIGPLAN Not.*, 42(6):32–41, 2007.
- [28] G. Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, and Z. Su. Dynamic test input generation for web applications. In *ISSTA '08: Proceedings of the 2008 international symposium on Software testing and analysis*, pages 249–260, New York, NY, USA, 2008. ACM.